Neural Machine Translation with Attention mechanism

What is Attention?

Attention is an interface between the encoder and decoder that provides the decoder with information from every encoder hidden state. With this setting, the model is able to selectively focus on useful parts of the input sequence and hence, learn the alignment between them. This helps the model to cope effectively with long input sentences .

```
!pip install chart-studio
!pip install kagglehub
```

```
Collecting chart-studio
  Downloading chart_studio-1.1.0-py3-none-any.whl (64 kB)
     |████████████████████████████████| 64 kB 1.9 MB/s
Requirement already satisfied: requests in /opt/conda/lib/python3.7/site-packages (from chart-studio) (2.23.0)
Requirement already satisfied: six in /opt/conda/lib/python3.7/site-packages (from chart-studio) (1.14.0)
Requirement already satisfied: retrying>=1.3.3 in /opt/conda/lib/python3.7/site-packages (from chart-studio) (1.3.3)
Requirement already satisfied: plotly in /opt/conda/lib/python3.7/site-packages (from chart-studio) (4.14.0)
Requirement already satisfied: retrying>=1.3.3 in /opt/conda/lib/python3.7/site-packages (from chart-studio) (1.3.3)
Requirement already satisfied: six in /opt/conda/lib/python3.7/site-packages (from chart-studio) (1.14.0)
Requirement already satisfied: chardet<4,>=3.0.2 in /opt/conda/lib/python3.7/site-packages (from requests->chart-studio) (3.0.4)
Requirement already satisfied: urllib3!=1.25.0,!=1.25.1,<1.26,>=1.21.1 in /opt/conda/lib/python3.7/site-packages (from requests->chart-studio) (1.25.9)
Requirement already satisfied: idna<3,>=2.5 in /opt/conda/lib/python3.7/site-packages (from requests->chart-studio) (2.9)
Requirement already satisfied: certifi>=2017.4.17 in /opt/conda/lib/python3.7/site-packages (from requests->chart-studio) (2020.12.5)
Requirement already satisfied: six in /opt/conda/lib/python3.7/site-packages (from chart-studio) (1.14.0)
Installing collected packages: chart-studio
Successfully installed chart-studio-1.1.0
WARNING: You are using pip version 20.3.1; however, version 24.0 is available.
You should consider upgrading via the '/opt/conda/bin/python3.7 -m pip install --upgrade pip' command.
Collecting kagglehub
  Downloading kagglehub-0.2.9-py3-none-any.whl (39 kB)
Requirement already satisfied: requests in /opt/conda/lib/python3.7/site-packages (from kagglehub) (2.23.0)
Requirement already satisfied: tqdm in /opt/conda/lib/python3.7/site-packages (from kagglehub) (4.45.0)
Requirement already satisfied: packaging in /opt/conda/lib/python3.7/site-packages (from kagglehub) (20.1)
Requirement already satisfied: six in /opt/conda/lib/python3.7/site-packages (from packaging->kagglehub) (1.14.0)
Requirement already satisfied: pyparsing>=2.0.2 in /opt/conda/lib/python3.7/site-packages (from packaging->kagglehub) (2.4.7)
Requirement already satisfied: chardet<4,>=3.0.2 in /opt/conda/lib/python3.7/site-packages (from requests->kagglehub) (3.0.4)
Requirement already satisfied: urllib3!=1.25.0,!=1.25.1,<1.26,>=1.21.1 in /opt/conda/lib/python3.7/site-packages (from requests->kagglehub) (1.25.9)
Requirement already satisfied: idna<3,>=2.5 in /opt/conda/lib/python3.7/site-packages (from requests->kagglehub) (2.9)
Requirement already satisfied: certifi>=2017.4.17 in /opt/conda/lib/python3.7/site-packages (from requests->kagglehub) (2020.12.5)
Installing collected packages: kagglehub
Successfully installed kagglehub-0.2.9
WARNING: You are using pip version 20.3.1; however, version 24.0 is available.
You should consider upgrading via the '/opt/conda/bin/python3.7 -m pip install --upgrade pip' command.
```

```
import pandas as pd
import tensorflow as tf
import matplotlib.pyplot as plt
from sklearn.model_selection import train_test_split

import unicodedata
import re
import numpy as np
import time
import string

from plotly.offline import init_notebook_mode, iplot
import plotly.graph_objs as go

import os
for dirname, _, filenames in os.walk('/kaggle/input'):
    for filename in filenames:
        print(os.path.join(dirname, filename))
```

```
/kaggle/input/quran-dataset-english-and-urdu/Quran-UR (1)
/kaggle/input/quran-dataset-english-and-urdu/Quran-EN (1)
```

As in case of any NLP task, after reading the input file, we perform the basic cleaning and preprocessing as follows:

The Dataset : We need a dataset that contains English sentences and their Portuguese translations which can be freely downloaded from this link. Download the file fra-eng.zip and extract it. On each line, the text file contains an English sentence and its French translation, separated by a tab.

```
import kagglehub

# Download latest version
path = kagglehub.dataset_download("devilgb666/quran-dataset-english-and-urdu")

print("Path to dataset files:", path)
```

```
Path to dataset files: /kaggle/input/quran-dataset-english-and-urdu
```

```
with open('/kaggle/input/quran-dataset-english-and-urdu/Quran-EN (1)', 'r', encoding='utf-8') as file:
    english_text = file.read().splitlines()

with open('/kaggle/input/quran-dataset-english-and-urdu/Quran-UR (1)', 'r', encoding='utf-8') as file:
    urdu_text = file.read().splitlines()


print("Englist Text records: ",len(english_text), ', Urdu Text records: ' ,len(urdu_text))
```

```
Englist Text records:  6414 , Urdu Text records:  6414
```

```
exclude = set(string.punctuation) # Set of all special characters
remove_digits = str.maketrans('', '', string.digits) # Set of all digits


def preprocess_eng_sentence(sent):
    '''Function to preprocess English sentence'''
    sent = sent.lower() # lower casing
    sent = re.sub("'", '', sent) # remove the quotation marks if any
    sent = ''.join(ch for ch in sent if ch not in exclude)
    sent = sent.translate(remove_digits) # remove the digits
    sent = sent.strip()
    sent = re.sub(" +", " ", sent) # remove extra spaces
    sent = '<start> ' + sent + ' <end>' # add <start> and <end> tokens
    return sent


urdu_diacritics  = ['ِ', '□'', 'ٔ', 'ٍ', 'ً', 'َ']

def remove_diacritics(text):
    for letter in text:

        if letter in urdu_diacritics:
            text = text.replace(letter, '')
    return text




urdu_digits = ['۶', '۴', '۵', '۸', '۲', '۰', '۷', '۹', '۳', '۱']

english_digits=['1','2','3','4','5','6','7','8','9','0']

def remove_numbers(text):
    for letter in text:
        if letter in urdu_digits or letter in english_digits :
            text = text.replace(letter, '')
    return text


def preprocess_ur_sentence(sent):
  sent = re.sub("'", '', sent) # remove the quotation marks if any
  sent = remove_diacritics(sent)
  sent = re.sub(r'[ـ،؟!#@=+٪،ـ؛)(}{]', '',sent)
  sent = remove_numbers(sent)
  sent = sent.strip()
  sent = re.sub(" +", " ", sent) # remove extra spaces
  sent = '<start> ' + sent + ' <end>' # add <start> and <end> tokens
  return sent


sent_pairs = []
for eng, ur in zip(english_text, urdu_text):
    eng = preprocess_eng_sentence(eng)
    ur = preprocess_ur_sentence(ur)
    sent_pair = [eng, ur]
    sent_pairs.append(sent_pair)
sent_pairs[5000:5010]
```

```
[['<start> and before that he destroyed the people of nuh noah as well surely they were extremely wicked and exceedingly defiant <end>',
  '<start> اور اس سے پہلے قوم نوح کو بھی ہلاک کیا بیشک وہ بڑے ہی ظالم اور بڑے ہی سرکش تھے <end>'],
 ['<start> and he is the one who raised up the overturned towns of the people of lut lot and smashed them down <end>',
  '<start> اور قوم لوط کی الٹی ہوئی بستیوں کو اوپر اٹھا کر اسی نے نیچے دے پٹکا <end>'],
 ['<start> then that covered them which did cover i e the stones were rained on them <end>',
  '<start> پس ان کو ڈھانپ لیا جس نے ڈھانپ لیا یعنی پھر ان پر پتھروں کی بارش کر دی گئی <end>'],
 ['<start> so o man which of the favours of your lord will you doubt <end>',
  '<start> سو اے انسان تو اپنے پروردگار کی کن کن نعمتوں میں شک کرے گا <end>'],
 ['<start> this holy prophet blessings and peace be upon him is also a warner of the warners of old <end>',
  '<start> یہ رسول اکرم صلی اللہ علیہ وآلہ وسلم بھی اگلے ڈر سنانے والوں میں سے ایک ڈر سنانے والے ہیں <end>'],
 ['<start> the imminent hour of judgment has drawn near <end>',
  '<start> آنے والی قیامت کی گھڑی قریب آپہنچی <end>'],
 ['<start> no one except allah can bring it forth and establish <end>',
  '<start> اﷲ کے سوا اسے کوئی ظاہر اور قائم کرنے والا نہیں ہے <end>'],
 ['<start> so do you wonder at this revelation <end>',
  '<start> پس کیا تم اس کلام سے تعجب کرتے ہو <end>'],
 ['<start> and do you laugh and not weep <end>',
  '<start> اور تم ہنستے ہو اور روتے نہیں ہو <end>'],
 ['<start> while you are busy playing a game of negligence <end>',
  '<start> اور تم غفلت کی کھیل میں پڑے ہو <end>']]
```

```
# This class creates a word -> index mapping (e.g,. "dad" -> 5) and vice-versa
# (e.g., 5 -> "dad") for each language,
class LanguageIndex():
    def __init__(self, lang):
        self.lang = lang
        self.word2idx = {}
        self.idx2word = {}
        self.vocab = set()

        self.create_index()

    def create_index(self):
        for phrase in self.lang:
            self.vocab.update(phrase.split(' '))

        self.vocab = sorted(self.vocab)

        self.word2idx['<pad>'] = 0
        for index, word in enumerate(self.vocab):
            self.word2idx[word] = index + 1

        for word, index in self.word2idx.items():
            self.idx2word[index] = word


def max_length(tensor):
    return max(len(t) for t in tensor)
```

Tokenization and Padding

```
def load_dataset(pairs, num_examples):
    # pairs => already created cleaned input, output pairs

    # index language using the class defined above
    inp_lang = LanguageIndex(en for en, ma in pairs)
    targ_lang = LanguageIndex(ma for en, ma in pairs)

    # Vectorize the input and target languages

    # English sentences
    input_tensor = [[inp_lang.word2idx[s] for s in en.split(' ')] for en, ma in pairs]

    # Marathi sentences
    target_tensor = [[targ_lang.word2idx[s] for s in ma.split(' ')] for en, ma in pairs]

    # Calculate max_length of input and output tensor
    # Here, we'll set those to the longest sentence in the dataset
    max_length_inp, max_length_tar = max_length(input_tensor), max_length(target_tensor)

    # Padding the input and output tensor to the maximum length
    input_tensor = tf.keras.preprocessing.sequence.pad_sequences(input_tensor,
                                                                 maxlen=max_length_inp,
                                                                 padding='post')

    target_tensor = tf.keras.preprocessing.sequence.pad_sequences(target_tensor,
                                                                  maxlen=max_length_tar,
                                                                  padding='post')

    return input_tensor, target_tensor, inp_lang, targ_lang, max_length_inp, max_length_tar
```

```
input_tensor, target_tensor, inp_lang, targ_lang, max_length_inp, max_length_targ = load_dataset(sent_pairs, len(english_text))
```

Creating training and validation sets using an 80-20 split

```
# Creating training and validation sets using an 80-20 split
input_tensor_train, input_tensor_val, target_tensor_train, target_tensor_val = train_test_split(input_tensor, target_tensor, test_size=0.1, random_state = 101)

# Show length
len(input_tensor_train), len(target_tensor_train), len(input_tensor_val), len(target_tensor_val)
```

```
(5772, 5772, 642, 642)
```

```
BUFFER_SIZE = len(input_tensor_train)
BATCH_SIZE = 4
N_BATCH = BUFFER_SIZE//BATCH_SIZE
```

```
embedding_dim = 32
units = 64
vocab_inp_size = len(inp_lang.word2idx)
vocab_tar_size = len(targ_lang.word2idx)

dataset = tf.data.Dataset.from_tensor_slices((input_tensor_train, target_tensor_train)).shuffle(BUFFER_SIZE)
dataset = dataset.batch(BATCH_SIZE, drop_remainder=True)
```

We'll be using GRUs instead of LSTMs as we only have to create one state and implementation would be easier.

Create GRU units

```
def gru(units):

    return tf.keras.layers.GRU(units,
                               return_sequences=True,
                               return_state=True,
                               recurrent_activation='sigmoid',
                               recurrent_initializer='glorot_uniform')
```

The next step is to define the encoder and decoder network.

The input to the encoder will be the sentence in English and the output will be the hidden state and cell state of the GRU.

```
class Encoder(tf.keras.Model):
    def __init__(self, vocab_size, embedding_dim, enc_units, batch_sz):
        super(Encoder, self).__init__()
        self.batch_sz = batch_sz
        self.enc_units = enc_units
        self.embedding = tf.keras.layers.Embedding(vocab_size, embedding_dim)
        self.gru = tf.keras.layers.GRU(self.enc_units, return_sequences=True, return_state=True)

    def call(self, x, hidden):
        x = self.embedding(x)
        output, state = self.gru(x, initial_state=hidden)
        return output, state


    def initialize_hidden_state(self):
        return tf.zeros((self.batch_sz, self.enc_units))
```

The next step is to define the decoder. The decoder will have two inputs: the hidden state and cell state from the encoder and the input sentence, which actually will be the output sentence with a token appended at the beginning.

```
class Decoder(tf.keras.Model):
    def __init__(self, vocab_size, embedding_dim, dec_units, batch_sz):
        super(Decoder, self).__init__()
        self.batch_sz = batch_sz
        self.dec_units = dec_units
        self.embedding = tf.keras.layers.Embedding(vocab_size, embedding_dim)
        self.gru = tf.keras.layers.GRU(self.dec_units, return_sequences=True, return_state=True)
        self.fc = tf.keras.layers.Dense(vocab_size)

        # Used for attention
        self.W1 = tf.keras.layers.Dense(self.dec_units)
        self.W2 = tf.keras.layers.Dense(self.dec_units)
        self.V = tf.keras.layers.Dense(1)

    def call(self, x, hidden, enc_output):
        hidden_with_time_axis = tf.expand_dims(hidden, 1)

        # Score shape == (batch_size, max_length, 1)
        # We get 1 at the last axis because we are applying tanh(FC(EO) + FC(H)) to self.V
        score = self.V(tf.nn.tanh(self.W1(enc_output) + self.W2(hidden_with_time_axis)))

        # Attention weights shape == (batch_size, max_length, 1)
        attention_weights = tf.nn.softmax(score, axis=1)

        # Context vector shape after sum == (batch_size, hidden_size)
        context_vector = attention_weights * enc_output
        context_vector = tf.reduce_sum(context_vector, axis=1)

        # X shape after passing through embedding == (batch_size, 1, embedding_dim)
        x = self.embedding(x)

        # X shape after concatenation == (batch_size, 1, embedding_dim + hidden_size)
        x = tf.concat([tf.expand_dims(context_vector, 1), x], axis=-1)

        # Passing the concatenated vector to the GRU
        output, state = self.gru(x)

        # Output shape == (batch_size * 1, hidden_size)
        output = tf.reshape(output, (-1, output.shape[2]))

        # Output shape == (batch_size * 1, vocab)
        x = self.fc(output)

        return x, state, attention_weights




    def initialize_hidden_state(self):
        return tf.zeros((self.batch_sz, self.dec_units))
```

Create encoder and decoder objects from their respective classes.

```
encoder = Encoder(vocab_inp_size, embedding_dim, units, BATCH_SIZE)
decoder = Decoder(vocab_tar_size, embedding_dim, units, BATCH_SIZE)
```

Define the optimizer and the loss function.

```
optimizer = tf.optimizers.Adam()

def loss_function(real, pred):
    mask = 1 - np.equal(real, 0)
    loss_ = tf.nn.sparse_softmax_cross_entropy_with_logits(labels=real, logits=pred) * mask
    return tf.reduce_mean(loss_)
```

```
checkpoint_dir = './training_checkpoints'
checkpoint_prefix = os.path.join(checkpoint_dir, "ckpt")
checkpoint = tf.train.Checkpoint(optimizer=optimizer,
                                 encoder=encoder,
                                 decoder=decoder)
```

Training the Model

```
EPOCHS = 5

for epoch in range(EPOCHS):
    start = time.time()

    hidden = encoder.initialize_hidden_state()
    total_loss = 0

    for (batch, (inp, targ)) in enumerate(dataset):
        loss = 0

        with tf.GradientTape() as tape:
            enc_output, enc_hidden = encoder(inp, hidden)

            dec_hidden = enc_hidden

            dec_input = tf.expand_dims([targ_lang.word2idx['<start>']] * BATCH_SIZE, 1)

            # Teacher forcing - feeding the target as the next input
            for t in range(1, targ.shape[1]):
                # passing enc_output to the decoder
                predictions, dec_hidden, _ = decoder(dec_input, dec_hidden, enc_output)

                loss += loss_function(targ[:, t], predictions)
                # using teacher forcing
                dec_input = tf.expand_dims(targ[:, t], 1)

        batch_loss = (loss / int(targ.shape[1]))

        total_loss += batch_loss

        variables = encoder.variables + decoder.variables

        gradients = tape.gradient(loss, variables)

        optimizer.apply_gradients(zip(gradients, variables))

        if batch % 100 == 0:
            print('Epoch {} Batch {} Loss {:.4f}'.format(epoch + 1,
                                                         batch,
                                                         batch_loss.numpy()))
    # saving (checkpoint) the model every epoch
    checkpoint.save(file_prefix = checkpoint_prefix)

    print('Epoch {} Loss {:.4f}'.format(epoch + 1,
                                        total_loss / N_BATCH))
    print('Time taken for 1 epoch {} sec\n'.format(time.time() - start))
```

```
Epoch 4 Batch 700 Loss 0.5838
Epoch 4 Batch 700 Loss 0.5838
Epoch 4 Batch 800 Loss 1.2043
Epoch 4 Batch 900 Loss 0.7674
Epoch 4 Batch 1000 Loss 0.5193
Epoch 4 Batch 1100 Loss 0.8832
Epoch 4 Batch 1200 Loss 0.7733
Epoch 4 Batch 1300 Loss 0.5060
Epoch 4 Batch 1400 Loss 0.5172
Epoch 4 Loss 0.7655
Time taken for 1 epoch 3241.0406532287598 sec

Epoch 5 Batch 0 Loss 1.0413
Epoch 5 Batch 100 Loss 1.0431
Epoch 5 Batch 200 Loss 0.7684
Epoch 5 Batch 300 Loss 0.8745
Epoch 5 Batch 400 Loss 0.8078
Epoch 5 Batch 500 Loss 0.7376
Epoch 5 Batch 600 Loss 1.0140
Epoch 5 Batch 700 Loss 0.3181
Epoch 5 Batch 800 Loss 0.8877
Epoch 5 Batch 900 Loss 0.2860
Epoch 5 Batch 1000 Loss 0.4718
Epoch 5 Batch 1100 Loss 0.6838
Epoch 5 Batch 1200 Loss 0.2796
Epoch 5 Batch 1300 Loss 0.5770
Epoch 5 Batch 1400 Loss 0.9555
Epoch 5 Loss 0.7367
Time taken for 1 epoch 3244.877701282501 sec
```

Restoring the latest checkpoint

```
# restoring the latest checkpoint in checkpoint_dir
checkpoint.restore(tf.train.latest_checkpoint(checkpoint_dir))
```

```
<tensorflow.python.training.tracking.util.CheckpointLoadStatus at 0x7df412471490>
```

Inference setup and testing:

```
def evaluate(inputs, encoder, decoder, inp_lang, targ_lang, max_length_inp, max_length_targ):

    attention_plot = np.zeros((max_length_targ, max_length_inp))
    sentence = ''
    for i in inputs[0]:
        if i == 0:
            break
        sentence = sentence + inp_lang.idx2word[i] + ' '
    sentence = sentence[:-1]

    inputs = tf.convert_to_tensor(inputs)
    result = ''

    hidden = [tf.zeros((1, units))]
    enc_out, enc_hidden = encoder(inputs, hidden)

    dec_hidden = enc_hidden
    dec_input = tf.expand_dims([targ_lang.word2idx['<start>']], 0)

    for t in range(max_length_targ):
        predictions, dec_hidden, attention_weights = decoder(dec_input, dec_hidden, enc_out)

        # storing the attention weights to plot later on
        attention_weights = tf.reshape(attention_weights, (-1, ))
        attention_plot[t] = attention_weights.numpy()

        predicted_id = tf.argmax(predictions[0]).numpy()

        result += targ_lang.idx2word[predicted_id] + ' '

        if targ_lang.idx2word[predicted_id] == '<end>':
            return result, sentence, attention_plot

        # the predicted ID is fed back into the model
        dec_input = tf.expand_dims([predicted_id], 0)

    return result, sentence, attention_plot
```

Function to predict (translate) a randomly selected test point

```
def predict_random_val_sentence():
    actual_sent = ''
    k = np.random.randint(len(input_tensor_val))
    random_input = input_tensor_val[k]
    random_output = target_tensor_val[k]
    random_input = np.expand_dims(random_input,0)
    result, sentence, attention_plot = evaluate(random_input, encoder, decoder, inp_lang, targ_lang, max_length_inp, max_length_targ)
    print('Input: {}'.format(sentence[8:-6]))
    print('Predicted translation: {}'.format(result[:-6]))
    for i in random_output:
        if i == 0:
            break
        actual_sent = actual_sent + targ_lang.idx2word[i] + ' '
    actual_sent = actual_sent[8:-7]
    print('Actual translation: {}'.format(actual_sent))
    attention_plot = attention_plot[:len(result.split(' '))-2, 1:len(sentence.split(' '))-1]
    sentence, result = sentence.split(' '), result.split(' ')
    sentence = sentence[1:-1]
    result = result[:-2]
```

```
predict_random_val_sentence()
```

```
Input: these which we read out to you are signs and the wisdom laden advice
Predicted translation: اور ان کے لئے کہ وہ لوگ ہیں
Actual translation: یہ جو ہم آپ کو پڑھ کر سناتے ہیں یہ نشانیاں ہیں اور حکمت والی نصیحت ہے
```

```
predict_random_val_sentence()
```

```
Input: and the holy messenger blessings and peace be upon him will submit o lord surely my people had utterly abandoned this quran
Predicted translation: ے لئے اس کے لئے اس کے لئے اس کے لئے اس کے لئے اس کے لئے اس کے لئے اس کے لئے اس کے لئے اس کے لئے اس کے لئے اس کے لئے اس کے لئے اس کے ل
Actual translation: اور رسول اکرم صلی اللہ علیہ وآلہ وسلم عرض کریں گے : اے رب بیشک میری قوم نے اس قرآن کو بالکل ہی چھوڑ رکھا تھا
```

```
predict_random_val_sentence()
```

```
Input: and someone whom we give long life we cause him to degenerate towards childhood and debility in strength and disposition so do they not have sense
Predicted translation: ن کے لئے ان کے لئے ان کے لئے ان کے لئے ان کے لئے ان کے لئے ان کے لئے ان کے لئے ان کے لئے ان کے لئے ان کے لئے ان کے لئے ان کے لئے ان ک
Actual translation: اور ہم جسے طویل عمر دیتے ہیں اسے قوت و طبیعت میں واپس بچپن یا کمزوری کی طرف پلٹا دیتے ہیں پھر کیا وہ عقل نہیں رکھتے
```

```
predict_random_val_sentence()
```

```
Input: and surely the reward and the punishment of actions are bound to take place
Predicted translation: اور وہ لوگ ہیں
Actual translation: اور بیشک اعمال کی جزا و سزا ضرور واقع ہو کر رہے گی
```

```
predict_random_val_sentence()
```

```
Input: and allah alone sends winds that raise and build up clouds then we drive it the cloud towards some dry and dead land to water it then with that we give l
Predicted translation: ئے اس کے لئے اس کے لئے اس کے لئے اس کے لئے اس کے لئے اس کے لئے اس کے لئے اس کے لئے اس کے لئے اس کے لئے اس کے لئے اس کے لئے اس کے لئے
Actual translation: بنجر بستی کی طرف سیرابی کے لئے لے جاتے ہیں پھر ہم اس کے ذریعے اس زمین کو اس کی مردنی کے بعد زندگی عطا کرتے ہیں اسی طرح مردوں کا جی اٹھنا ہوگا
```

```
# 1. Install & import
!pip install nltk
import nltk
from nltk.translate.bleu_score import corpus_bleu

# 2. Restore your latest checkpoint
checkpoint.restore(tf.train.latest_checkpoint(checkpoint_dir))

# 3. Helper to convert token sequences back to sentences
def tensor_to_sentence(seq, lang):
    # If it's a TF tensor, turn into NumPy
    if hasattr(seq, 'numpy'):
        seq = seq.numpy()
        seq = seq.numpy()
    tokens = []
    for idx in seq:
        # skip padding, start, end
        if idx in {
            lang.word2idx.get('<pad>'),
            lang.word2idx.get('<start>'),
            lang.word2idx.get('<end>')
        }:
            continue
        tokens.append(lang.idx2word[idx])
    return ' '.join(tokens)

# 4. Build reference & hypothesis lists
references = []
hypotheses = []

for inp, targ in zip(input_tensor_val, target_tensor_val):
    # a) reference: true target
    ref_sentence = tensor_to_sentence(targ, targ_lang)
    references.append([ref_sentence.split()])

    # b) model prediction (unpack only the decoded string)
    decoded, _, _ = evaluate(
        inp[None, :],      # add batch dimension
        encoder,
        decoder,
        inp_lang,
        targ_lang,
        max_length_inp,
        max_length_targ
    )
    hypotheses.append(decoded.strip().split())

# 5. Compute corpus BLEU
bleu_score = corpus_bleu(references, hypotheses) * 100
print(f'Corpus BLEU score on validation set: {bleu_score:.2f}')
```

```
Requirement already satisfied: nltk in /opt/conda/lib/python3.7/site-packages (3.2.4)
Requirement already satisfied: six in /opt/conda/lib/python3.7/site-packages (from nltk) (1.14.0)
WARNING: You are using pip version 20.3.1; however, version 24.0 is available.
You should consider upgrading via the '/opt/conda/bin/python3.7 -m pip install --upgrade pip' command.
Corpus BLEU score on validation set: 0.16
```

Start coding or generate with AI.